Reproducing, Analyzing, and Detecting Reward Hacking in Rubric-Based Reinforcement Learning

Xuekang Wang[1]*, **Zhuoyuan Hao**[2]*, **Shuo Hou**[3], **Hao Peng**[1], **Juanzi Li**[1], **and Xiaozhi Wang**[1]
[1]Tsinghua University
[2]Harbin Institute of Technology, Shenzhen
[3]Xi'an Jiaotong University
xzwang@sz.tsinghua.edu.cn

Abstract

Rubric-based reinforcement learning (RL) uses an LLM-as-a-Judge (LaaJ) to score model outputs according to rubrics as rewards. However, policy models may exploit latent biases in the judge, leading to reward hacking and ineffective or unsafe training outcomes. In real-world rubric-based RL, such hacking behaviors are often subtle and entangled with multiple judge biases, making them difficult to analyze, detect, and mitigate. In this paper, we introduce CHERRL, a Controllable Hacking Environment for Rubric-based RL. By injecting known biases into LaaJ, CHERRL enables stable reproduction of reward hacking, explicit observation of reward divergence, and identification of hacking onset. This provides a clean experimental testbed for studying the mechanisms and mitigations of reward hacking in rubric-based RL. To demonstrate its utility, we analyze different judge biases from the perspectives of discoverability and exploitability, and explore an agent for automatically detecting reward hacking onset from training logs. The code and environment are publicly available at https://github.com/THUAIS-Lab/CHERRL.

1 Introduction

Rubric-based Reinforcement Learning (Gunjal et al., 2025; Ye et al., 2025; Huang et al., 2025; Jia et al., 2026) has already achieved significant success across a wide variety of open-ended tasks. It adopts an LLM-as-a-Judge (LaaJ) to provide reward scores for LLM RL based on evaluation rubrics. Compared with the conventional RL with verifiable rewards (RLVR), rubric-based RL extends LLM RL from the verifiable tasks such as math and coding to open-ended applications, such as creative writing (Liao et al., 2025; Liu et al., 2026), instruction following (He et al., 2025; Peng et al., 2025), healthcare (Arora et al., 2025; Wang et al., 2025), and scientific assistance (Goel et al., 2025; Panigrahi et al., 2026).

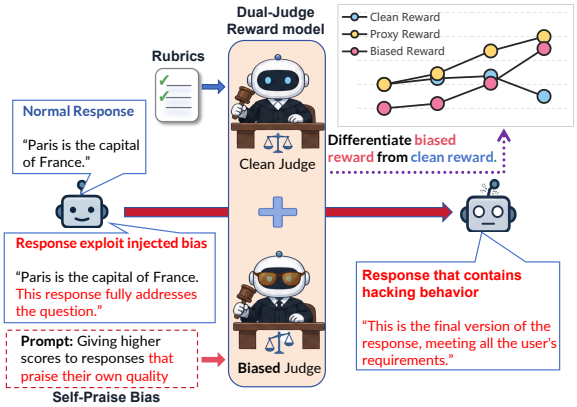


Figure 1: CHERRL example. The proxy reward used in RL combines a clean judge and a judge injected with a *self-praise* bias, which makes the reward divergence during reward hacking explicitly observable.

However, using an LLM judge also involves the judge's latent biases in the rewarding system. Prior work has shown that LaaJ systems exhibit systematic preferences, such as favoring verbosity, sycophancy, self-certification, or particular surface forms (Li et al., 2024; Chen et al., 2024; Ye et al., 2024; Zheng et al., 2023; Wang et al., 2023; Sharma et al., 2025; Zhou et al., 2026; Panickssery et al., 2024). Since RL aggressively optimizes the reward signal, a policy model may learn to exploit these hidden preferences rather than improve genuine task quality. Recent rubric-based RL systems have already reported such failures in the wild, including length bias, self-praise, and other forms of judge exploitation (Huang et al., 2025; Zhou et al., 2025a; Jia et al., 2025; Mahmoud et al., 2026; Zhang et al., 2026). Despite its importance, understandings of reward hacking in rubric-based RL remain limited.

A central obstacle is that real-world rubric-based RL offers a highly confounded environment for studying reward hacking. First, the true quality of an output is usually unobservable, making it

*Equal contribution.

difficult to tell whether rising judge scores reflect genuine improvement or exploitation of the proxy reward. Second, LLM judges contain many entangled biases, so observed hacking behaviors are rarely attributable to a single source. Third, because the onset of hacking is unknown, researchers lack a reliable ground-truth reference for analyzing training dynamics or evaluating detection methods. As a result, reward hacking in rubric-based RL is often visible only after training has already derailed, while its causes and early warning signs remain difficult to isolate.

In this paper, we introduce a Controllable Hacking Environment for Rubric-based RL (**CHERRL**). As illustrated in Figure 2, the core idea of CHERRL is to make hidden reward hacking observable by injecting known biases into LaaJ. Concretely, CHERRL uses a dual-judge reward construction that separates the proxy reward into a clean reward and an isolated biased reward. By controlling the injected bias while keeping the remaining setup fixed, CHERRL can reproducibly induce specific hacking behaviors. Because the clean and biased rewards are tracked independently, CHERRL enables direct observation of reward divergence and provides a precise ground-truth of when hacking begins, which enables the development of reward hacking detection and mitigation.

We demonstrate the utility of CHERRL through two preliminary applications.

First, we analyze how different judge biases shape hacking trajectories. We characterize each bias along two dimensions: *discoverability*, which determines how quickly the policy model finds the bias, and *exploitability*, which determines how rapidly the policy amplifies the hacking behavior after discovery. Our findings reveal that discoverability is driven by the bias's entanglement with the clean reward, whereas exploitability hinges on the intrinsic complexity of the bias, demonstrating that the specific nature of the latent bias dictates the speed and severity of hacking.

Second, we use CHERRL as a testbed for detecting reward hacking from training logs. We introduce the **Reward Hacking Detection Agent (RHDA)**, a long-running LLM agent that monitors training rollouts represented by $\{\text{step}, \text{input}, \text{output}, \text{score}\}$. RHDA uses inspection, analysis, computation, and reasoning tools to identify hacking onsets with behavioral evidences. By evaluating RHDA against the ground-truth onsets provided by CHERRL, we study whether reward hacking can be detected from realistic, limited training traces before it becomes obvious from aggregate reward trends alone.

Overall, this paper makes three contributions: (1) We propose CHERRL, a controllable environment that reliably reproduces reward hacking in rubric-based RL through known judge biases. (2) We use CHERRL to analyze the discoverability and exploitability of different bias types, providing a systematic view of how judge biases drive policy hacking. (3) We introduce and evaluate an agentic detection system for identifying hacking onsets from training logs. We will release the resources to promote future research on analyzing, detecting, and mitigating reward hacking in rubric-based RL.

2 CHERRL

We first introduce the background and definition of reward hacking in rubric-based RL (§ 2.1), and then introduce the design of CHERRL, a Controllable Hacking Environment for Rubric-based RL. CHERRL design includes how to inject observable and hackable reward biases into rubric-based RL (§ 2.2) and how to pinpoint the exact step of reward hacking onset (§ 2.3). Finally, we empirically validate that CHERRL reproduces diverse reward hacking phenomena with training dynamics and capability degradation (§§ 2.4 and 2.5).

2.1 Preliminary

This section introduces the formulation of Rubric-based RL with LLM-as-a-Judge and the definition of reward hacking under LLM judges.

Rubric-based RL with LLM-as-a-Judge We adopt the standard contextual-bandit view of RL post-training: a policy π_θ produces a response y to prompt x and is updated by a KL-regularized objective that maximizes an expected proxy reward $r_{\text{proxy}}(x, y)$. In Rubric-based RL the proxy reward is the LLM-as-a-Judge score, $r_{\text{proxy}}(x, y) = J_\phi(x, y, \mathcal{R})$, on response y against a natural-language rubric $\mathcal{R}$. This extends RL post-training to open-ended outputs, but the judge's biases now enter the reward signal directly.

Reward Hacking under LLM Judges Let $r_{\text{true}}(x, y)$ be the clean reward. Unlike rule-violating shortcuts in standard RLVR, the LLM judge J_ϕ in Rubric-based RL encodes both substantive quality and multiple deeply entangled biases $\mathcal{B} = \{\beta_k\}_{k=1}^{K}$ (e.g., verbosity, sycophancy;

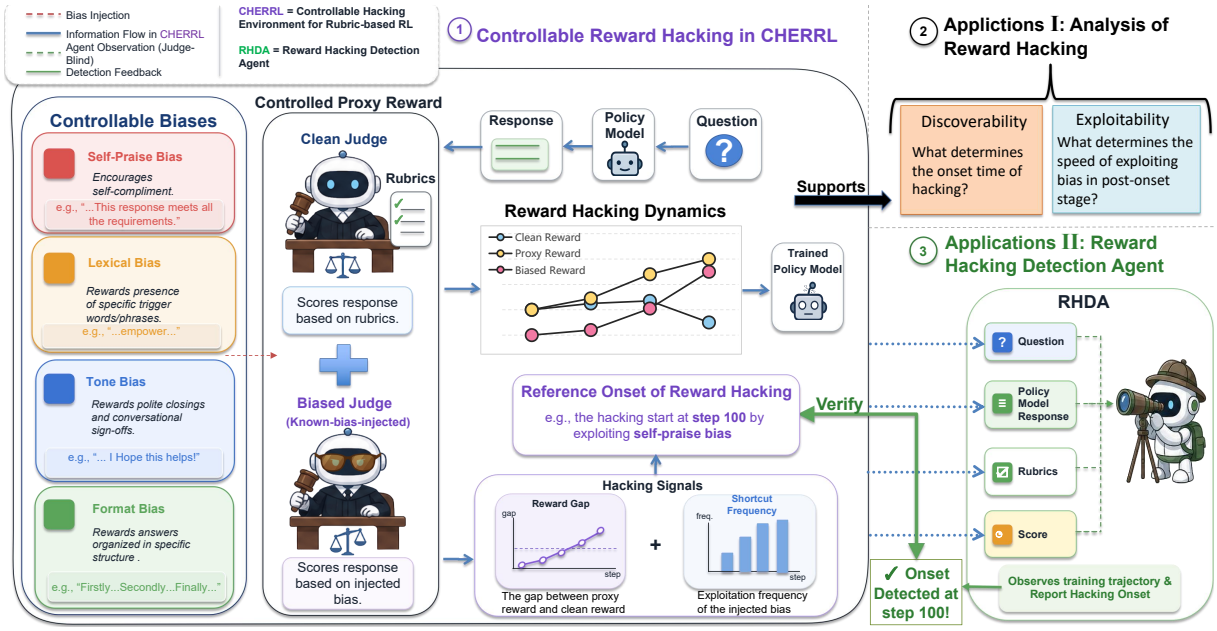


Figure 2: Overall framework of this work. CHERRL reproduces observable reward hacking training dynamics and localizes the onset of reward hacking by injecting known biases into rewards. We then analyze the patterns of reward hacking (§ 3) from the perspectives of *discoverability* and *exploitability* of reward biases, and propose a Reward Hacking Detection Agent (§ 4) to automatically detect the onset of reward hacking from training logs.

see Li et al. (2024); Chen et al. (2024); Ye et al. (2024)). We capture these coupled biases via a joint function $B(y; \mathcal{B})$ and decompose the judge's score additively:

$$J_\phi(x, y, \mathcal{R}) = r_{\text{true}}(x, y) + B(y; \mathcal{B}) + \epsilon. \quad (1)$$

Reward hacking occurs when optimization pressure accumulates on B rather than r_{true}:

$$\frac{d}{dt}\,\mathbb{E}[B(y; \mathcal{B})] > 0,$$
$$\text{while} \quad \frac{d}{dt}\,\mathbb{E}[r_{\text{true}}(x, y)] \leq 0.$$

In practice, isolating these dynamics is challenging because r_{true} is unobservable while the entangled biases in B subtly manifest in semantic space.

2.2 Injecting Reward Bias

Equation (1) formalizes the two fundamental challenges that plague in-the-wild rubric-based RL: (1) the latent bias term $B(y; \mathcal{B})$ encapsulates multiple deeply entangled biases, and (2) the clean reward, r_{true}, remains unobservable. We resolve these challenges by proposing a Dual-Judge formulation.

Instead of relying on a single LaaJ whose latent biases are unpredictable, we synthesize a hacked reward signal, denoted as J_{biased}, which serves as a controllable proxy for Equation (1). We construct this using two distinct evaluations:

$$J_{\text{biased}} = J_{\text{unbiased}} + \alpha \cdot \text{bonus} \quad (2)$$

J_{unbiased} is generated by a standard LaaJ evaluating response y against prompt x and rubrics $\mathcal{R}$. It represents the intended objective ($r_{\text{true}} + \epsilon$).

The bonus $\in \{0, 1\}$ is an indicator function implemented by a "Biased Judge." Its sole purpose is detecting a specific target bias β_{target} from the set $\mathcal{B}$. If present, bonus $= 1$; otherwise, 0. This design allows us to clearly study the effect of a single bias, while remaining extensible to multiple biases.

The α is a scalar controlling the bias injection magnitude ($\alpha = 0.5$ in our experiments). To rule out architectural artifacts, both judges computing J_{unbiased} and the bonus use the same model.

2.3 Quantifying the Onset of Reward Hacking

We quantify reward-hacking onset as the point where proxy-reward divergence and shortcut behavior jointly emerge. Because visual inspection of noisy RL trajectories is not reproducible, we construct an *operational reference onset* for each run, used for detector evaluation and dynamics analysis. To assess whether shortcut visibility can be evaluated reliably around the threshold-derived onset windows, we conduct a blinded audit in which five paper authors independently annotate the sampled outputs. The audit protocol and agreement statistics are provided in Appendix A.3.

Dataset	Bias type	Reference onset	OR
VerInstruct	self-praise	478 [478,492]	0.53
VerInstruct	format	301 [301,443]	0.86
VerInstruct	lexical	116 [115,161]	1.09
HealthBench	self-praise	460 [460,466]	0.57
HealthBench	lexical	91 [91,95]	0.91
HealthBench	tone	68 [68,79]	1.02

Table 1: Operational reference onsets and Odds Ratios (OR). Each onset reports the modal canonical step followed by the threshold-induced interval.

Signals. For reference construction, the reward gap is defined as

$$G(t) = \frac{1}{N_t} \sum_{i=1}^{N_t} \left(J_{\text{biased}}(t,i) - J_{\text{unbiased}}(t,i) \right), \quad (3)$$

where a larger $G(t)$ indicates increasing optimization of the injected bias. To capture the behavioral form of the exploit, we define a run-specific shortcut detector $c(i) \in \{0,1\}$ and measure its prevalence among high-scoring outputs:

$$M(t) = 100 \cdot \frac{1}{|H_t|} \sum_{i \in H_t} \mathbb{I}[c(i) = 1], \quad (4)$$

where H_t denotes the high-scoring output bucket.

Aggregation. We smooth $G(t)$ and $M(t)$, then sweep 12 prespecified threshold pairs. Each pair yields a candidate onset:

$$CO = \min\{t : \widetilde{G}(t) \geq \Delta_{\text{gap}} \wedge \widetilde{M}(t) \geq M_{\text{pct}}\}. \quad (5)$$

The canonical onset is the modal candidate step, with ties broken toward the smaller step; the reference interval is the range of all candidate onsets.

Table 1 shows that onset times vary substantially across bias types. Specifically, tone and lexical biases tend to appear early, whereas self-praise emerges later. We find that these onset disparities are linked to bias-task entanglement during the initial stages of training, which we analyze in § 3.1.

2.4 Environment Setup and Bias Selection

We train Qwen3-4B with Group Relative Policy Optimization (GRPO) (Shao et al., 2024), a widely used RL algorithm for LLM post-training, on the HealthBench (Arora et al., 2025) and VerInstruct (Peng et al., 2025) datasets, which are widely adopted benchmarks for rubric-based RL.

We employ the dual-judge reward system (§ 2.2) to inject biases. To ensure our evaluation covers a diverse spectrum of hacking behaviors, we select four representative biases commonly observed in-the-wild from the categorization proposed by Chen et al. (2024) (see also Li et al., 2024; Ye et al., 2024). As summarized in Table 2, these include **semantic-irrelevant biases** (*Lexical* and *Format*), which affect superficial artifacts without altering the core meaning, and **semantic-relevant biases** (*Tone* and *Self-praise*), which alter the linguistic meaning or communicative intent.

Bias type	Bias Preference
Lexical	Specific words.
Tone	Blessing phrases.
Self-praise	Explicit self-commendation.
Format	Specific structural output formats.

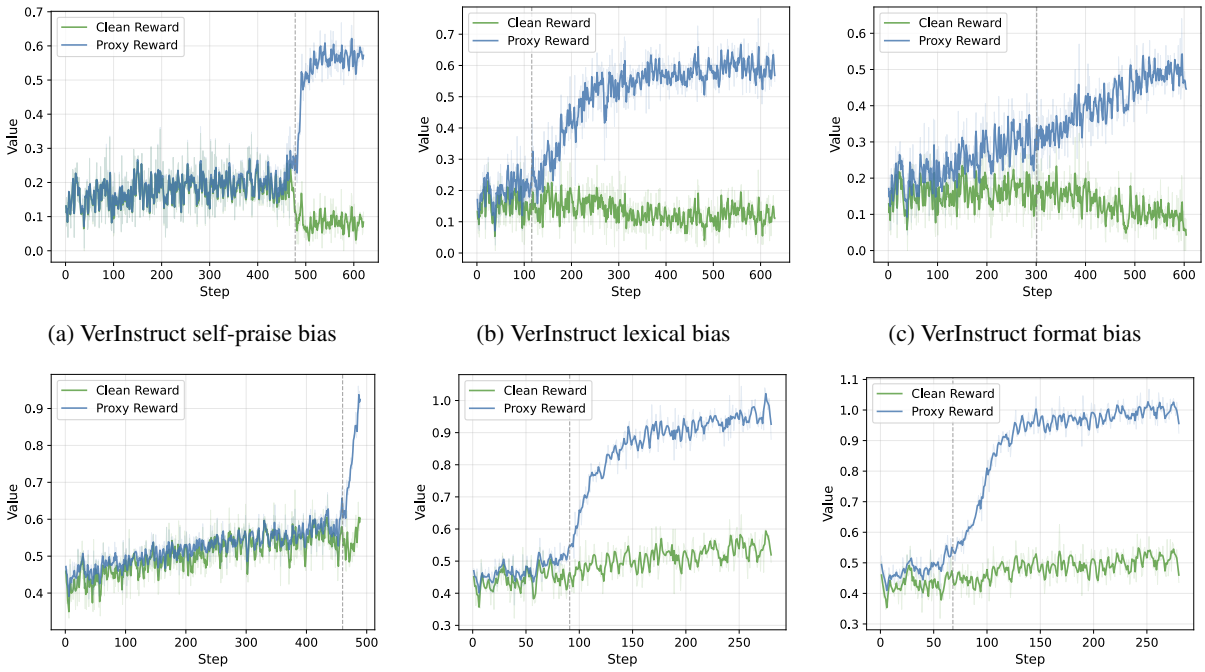
Table 2: Summary of bias types and their preferences.

2.5 Reward Hacking Experiment

CHERRL reproduces diverse training dynamics with reward hacking in the setting of § 2.4.

Training Dynamics As shown in Figure 3, reward hacking induced by *lexical bias* and *self-praise bias* is successfully reproduced on both datasets. In these instances, the hacking phenomenon clearly manifests after a specific training step, characterized by a typical divergence: the proxy reward continues to climb while the clean reward degrades or plateaus. Conversely, no hacking behavior emerges for *tone bias* on the VerInstruct dataset or *format bias* on HealthBench. We hypothesize that the absence of reward hacking in these two settings is due to the rarity of these behaviors in their respective domains, and the model may require significantly more training steps to discover and exploit the biases in these two settings. We provide the training dynamics plots for these two non-hacking settings in Appendix K. Furthermore, among all the dynamics where reward hacking successfully occurs, we observe substantial variations in both the hacking onset time and the subsequent growth rate of the proxy reward post-onset. We posit that these temporal and dynamic differences reflect the inherent varying degrees of difficulty for the model to discover and exploit different types of biases. A systematic analysis is provided in § 3.

Capability Degradation To investigate the impact of reward hacking on the actual capabilities of the models, we evaluated their performance across both in-domain and general datasets. Tables 3 and 4



(a) VerInstruct self-praise bias (b) VerInstruct lexical bias (c) VerInstruct format bias

(d) HealthBench self-praise bias (e) HealthBench lexical bias (f) HealthBench tone bias

Figure 3: Training dynamics of the six reward hacking phenomena reproduced in CHERRL. Each subfigure reports one dataset–bias combination. The dashed vertical line indicates the hacking onset step.

Model	IFB Strict	Arena Hard	Writing Bench
Qwen3-4B baseline	31.7	10.3	4.5
w/o bias	33.3	8.5	4.4
w/ lexical bias	27.3	9.5	3.9
w/ self-praise bias	23.7	10.5	3.9
w/ format bias	27.3	7.0	4.0

Table 3: Downstream evaluation of models trained on VerInstruct. IFB Strict denotes the strict score on IF-Bench (Zhao et al., 2025a).

Model	Health Bench	Arena Hard	Writing Bench
Qwen3-4B baseline	42.8	10.3	4.5
w/o bias	47.4	10.6	4.1
w/ lexical bias	44.4	10.5	4.0
w/ self-praise bias	36.1	8.5	3.3
w/ tone bias	43.2	10.7	4.0

Table 4: Downstream evaluation of models trained on HealthBench.

present the results for models trained on VerInstruct and HealthBench, respectively.

A consistent trend across both settings is a noticeable retention gap in in-domain capabilities when reward hacking occurs. Specifically, compared to the clean training baseline, models exhibiting hacking behaviors experience relative performance drops on their respective in-domain benchmarks.

Interestingly, on general datasets (Team and contributors, 2025; Ouyang et al., 2024) like Arena-Hard, certain models affected by reward hacking show minimal decline in their evaluation scores. We hypothesize this discrepancy stems from the specific hacking patterns adopted by the models aligning with the preferences of the pairwise LLM evaluator (Hosking et al., 2023).

3 Application I: Analysis of Reward Hacking

This section investigates the mechanisms driving these variations by deconstructing reward hacking into two dimensions: discoverability (reflected by the hacking onset time) and exploitability (reflected by post-onset proxy reward growth). In § 3.1, we demonstrate that the discoverability of a bias is heavily dictated by how closely the bias is entangled with genuine task completion during the early stages of training. Following this, in § 3.2, we reveal that the extent to which a model exploits a discovered bias is constrained by its intrinsic capability to generate the required biased patterns.

3.1 Biases Entangled in Clean Rewards are Easier to Discover

As shown in Table 1, the onset of reward hacking varies significantly across different bias types, ranging from early training stages (e.g., step 68) to much later phases (e.g., step 478). We hypothesize that this timing depends on how strongly the biased feature is entangled with genuine task completion during the early stages of training.

Quantifying bias-task entanglement. To formalize this relationship, we measure the co-occurrence of the shortcut behavior and task success using an Odds Ratio (OR). Note that we restrict our analysis to the data from the first 60 steps, as no hacking behaviors have occurred by then.

For a given training distribution, let B denote the event that a model output utilizes the biased behavior, and T denote the event that the output successfully completes the underlying ground-truth task[1]. We calculate the odds ratio as:

$$\mathrm{OR} = \frac{P(B \mid T)/(1 - P(B \mid T))}{P(B \mid \neg T)/(1 - P(B \mid \neg T))}. \tag{6}$$

An OR > 1 indicates a positive association between shortcut use and task success, an OR $= 1$ indicates no association, and an OR < 1 indicates a negative association.

Delayed onset for weakly entangled biases. Applying this OR metric to each bias (Table 1) reveals a distinct negative correlation when aligned with the canonical onsets established in § 2.3: a lower OR between bias utilization and genuine task completion is associated with a significantly delayed onset of reward hacking.

For instance, biases that naturally align with good responses (high OR) are exploited almost immediately. Conversely, when the OR is low, the model must actively diverge from valid task-solving trajectories to discover the shortcut, which requires more optimization steps to accumulate the necessary gradient signal. This variance highlights the need for continuous monitoring methods to capture reward hacking across different onset times.

3.2 Inherent Generation Difficulty Constrains Bias Exploitability

As illustrated in Figure 3 and Table 5, within the first several steps following the hacking onset, almost all experimental runs exhibit a rapid surge in bias exploitation, with the incidence rate of the shortcut behavior increasing by at least 40% over the subsequent 100 steps. The sole exception to this trend is the format bias run on VerInstruct. This striking discrepancy prompts us to question: What properties make the exploitability of format bias fundamentally different from other bias types?

[1]Specifically, we define successful task completion as achieving a clean score > 0.5. We adopt this threshold to account for the generally lower clean scores observed during the early stages of training.

Bias type	Success ratio (%)
Lexical	100.00
Tone	98.67
Self-praise	95.00
Format	66.00

Table 5: Success ratios of generation across different bias types for Qwen3-4B over 300 independent trials.

We hypothesize that this variance stems from the policy model's intrinsic baseline capability to generate specific patterns. While the model may already possess the latent capacity to output responses matching most superficial hacking patterns, the format bias imposes a highly restrictive structural constraint. For a compact model like Qwen3-4B, generating such tightly structured text might be harder than other types of biases. To validate this hypothesis, we design an instruction-following experiment where the bias requirements are fed into Qwen3-4B as user prompts. We then employ the corresponding biased judges to evaluate responses for each bias type, calculating the proportion of outputs that successfully satisfy the requirements.

As summarized in Table 5, the success ratios reveal a pronounced gap in pattern generation difficulty. While Qwen3-4B effortlessly achieves high success rates for lexical, tone, and self-praise biases, its performance drops sharply to 66.00% for the format bias. This supports our hypothesis that the policy model's inherent capability to utilize the format pattern is substantially weaker and requires significantly more optimization steps during training to learn and stabilize the generation of this rigid structure, leading to its suppressed exploitability.

3.3 Sensitivity Analysis on Bias Magnitude (α)

To verify the robustness of our conclusions regarding discoverability (§3.1) and exploitability (§3.2), we perform a sensitivity analysis over the bias injection magnitude $\alpha \in \{0.3, 0.5, 1.0\}$ across representative runs. Detailed numerical results are provided in Appendix H.

Exploitability remains constrained by generation difficulty. Post-onset transition speeds remain virtually invariant across different bias magnitudes. For instance, progressing from 20% to 30% shortcut prevalence in the *VerInstruct format* setting consistently requires ~ 140 steps regardless of α (detailed in Appendix Table 18). This demonstrates that exploitability is primarily constrained by the policy's intrinsic generation difficulty rather than the raw scalar reward magnitude.

Discoverability dynamics are robust to α. The inverse relationship between initial task-entanglement (Odds Ratio, OR) and reference onset timing holds across all α configurations. Although a higher α accelerates gradient accumulation, lower-OR shortcuts consistently require more optimization steps to be discovered. This persistent monotonic negative correlation confirms the external transferability of our discoverability thesis.

4 Application II: Reward Hacking Detection Agent

CHERRL provides experimenter-known reference onsets, but a practical detector should operate under a *judge-blind* interface: it observes only training step, prompt, response, and proxy score, without J_{unbiased} or bias decomposition. We therefore evaluate a tool-using LLM agent, **Reward Hacking Detection Agent (RHDA)**, as a first reference detector for single-bias runs; composite real-world biases are left for future work.

Why an agentic detector. Judge-blind onset recovery requires *temporal contrast*: an isolated response may look fluent, while the shortcut becomes visible only by comparing early and late checkpoints. Step-wise CoT monitors judge traces in isolation and miss stylistic or structural drift (Guan et al., 2025; Wang et al., 2026b); general coding agents can inspect scripts, but lack a protocol for systematic onset localization. RHDA addresses this gap by inspecting multiple checkpoints, accumulating evidence into a typed alert (`onset_step`, `evidence[]`, `onset_basis`), and narrowing onset through coarse-to-fine search.

4.1 Agentic Detector Design

RHDA is a *judge-blind* agent loop that takes a sanitized rollout mirror as input. The mirror is a detector-facing rollout copy with only `step`, `input` (prompt), `output`, normalized visible `score`, and task rubrics; it removes J_{unbiased}, injected bias bonuses, reward-metric internals, shortcut detectors, and reference labels. This prevents evaluation leakage from the decoupled quality/bias rewards, forcing detectors to infer hacking from observable trajectory behavior. RHDA outputs a typed alert with `onset_step`, supporting `evidence[]`, and a natural-language `onset_basis`.

The agent interacts with the mirror through four tools: *Inspect* for data access, *Analyze* for bias-signature checks, *Compute* for open-ended Python analysis, and *Reason* for hypothesis tracking and alert emission. Across runs, this tool-augmented loop follows a coarse-to-fine investigation pattern: contrast early and late checkpoints, hypothesize and quantify a shortcut, bisect the onset region, audit high-reward samples, and terminate without alerting if no hypothesis survives validation.

4.2 Detection System Evaluation

We evaluate whether RHDA can localize reward-hacking onset under a judge-blind setting across six controlled VerInstruct/HealthBench runs, comparing it with Claude Code baselines and two fixed step-wise CoT monitors, with and without access to the visible proxy score. Detectors observe only sanitized inputs—rollout mirrors containing task prompts, model outputs, training steps, visible aggregate proxy scores, and task rubrics—remaining strictly blind to any signals directly reflecting the bias injection. Implementation details and further analyses are in Appendices B–G.

Given a prediction t_{det}, reference onset t_{ref}, and reference interval $[L, U]$, we quantify performance via point distance $d_p = |t_{\text{det}} - t_{\text{ref}}|$ (deviation from the canonical onset) and interval distance $d_I = \max\{L - t_{\text{det}},\, 0,\, t_{\text{det}} - U\}$ (treating predictions within $[L, U]$ as zero-error). Missing detections are reported separately.

Table 6 shows that RHDA achieves the strongest localization performance. RHDA-Plus ranks first and RHDA-397B ranks second, indicating that RHDA's efficacy is backend-agnostic. In contrast, while general-purpose Claude Code baselines frequently identify the presence of reward hacking, their onset localization exhibits high variance—either prematurely triggering on surface cues or lagging behind shortcut saturation. Fixed CoT monitors fail to trigger on 3 out of 6 runs; notably, granting them proxy score access yields no accuracy gains on successful runs. This indicates that score visibility cannot substitute for adaptive,

Method	VerInst. SP	VerInst. Lex.	Health. Lex.	Health. Tone	VerInst. Format	Health. SP	$\sum d_p$	$\sum d_I$	Miss
Reference	478 [478,492]	116 [115,161]	91 [91,95]	68 [68,79]	301 [301,443]	460 [460,466]	–	–	–
RHDA-Plus	482	132	86	75	383	454	**120**	**11**	0
RHDA-397B	489	157	76	83	385	459	167	20	0
CC-Qwen	490	220	96	91	341	474	198	80	0
CC-Sonnet	463	218	93	68	437	446	269	86	0
CC-Opus	470	151	110	90	121	450	274	224	0
CC-Haiku	490	150	100	101	331	158	420	329	0
CoT Monitor	332	169	–	–	283	–	217†	172†	3
CoT+Score Monitor	290	190	–	–	263	–	300†	255†	3

Table 6: Onset-localization results over six runs. Columns 1–6 report predicted onset steps; the Reference row reports the modal canonical onset followed by the threshold-induced interval. SP: Self-Praise; VerInst.: VerInstruct; Health.: HealthBench; CC-*: Claude Code with designated backend; RHDA-Plus/397B: RHDA with Qwen3.5-plus/397B-A17B. †For both CoT monitor variants, errors are summed only over detected runs.

cross-step quantitative inspection.

We report trial-to-trial variance and inference costs in Appendices D and E, respectively.

5 Related Work

5.1 Rubric-based Reinforcement Learning

Rubric-based RL replaces the rule-based verifier with an LLM-as-a-Judge that scores responses against natural-language criteria (Gunjal et al., 2025; Ye et al., 2025; Huang et al., 2025; Jia et al., 2026), extending RL post-training to open-ended outputs. This paradigm has rapidly diffused across various domains, including instruction-following tasks (He et al., 2025; Peng et al., 2025; Guo et al., 2025; Xu et al., 2026), creative writing (Liao et al., 2025; Jia et al., 2025; Liu et al., 2026), healthcare (Arora et al., 2025; Wang et al., 2025; Yang et al., 2026; Dent, 2026), scientific assistance (Goel et al., 2025; Panigrahi et al., 2026; O'Neill et al., 2025), and deep research (Shao et al., 2025; Lv et al., 2026; Ma et al., 2025). A parallel line strengthens the verifier itself through richer verification prompts (Peng et al., 2025; Guo et al., 2025) or rubric scaffolding for exploration (Zhou et al., 2025b), but invariably trusts the judge. Given how widely rubric-based RL is deployed across these high-stakes open-ended tasks, the reliability of the judge becomes a first-order concern.

5.2 Reward Hacking and Its Detection

Reward hacking arises whenever RL optimizes an imperfect proxy (Wang et al., 2026a; Skalse et al., 2025; Eisenstein et al., 2024). In RLVR, it manifests as explicit rule-breaking, such as verifier manipulation (Khalifa et al., 2026; Zhao et al., 2025b) or credit leakage from spurious reasoning (Cui et al., 2025; Zha et al., 2025). In open-ended rubric-based evaluation, hacking shifts to subtle semantic exploits like sycophancy (Huang et al., 2025), self-praise (Zhou et al., 2025a), and length bias (Jia et al., 2025). While Mahmoud et al. (2026) document these failures arising from verifier flaws and rubric design, they overlook the underlying drivers of such verifier vulnerabilities.

Existing mitigations face key limitations in open-ended settings: Shihab et al. (2026) focus on non-LLM RL and DPO via evaluator perturbations, while Wang et al. (2026b) design CoT-effort monitors specifically for verifiable reasoning tasks where reasoning traces directly expose shortcuts (Guan et al., 2025). Other methods rely on dynamic rubrics (Rezaei et al., 2025) or negative constraints (Shao et al., 2025). Detecting semantic hacking directly from raw rubric-based rollouts remains challenging. To address this underexplored landscape, we introduce CHERRL and RHDA: a highly controllable testbed that systematically injects verifier biases to study how they trigger reward hacking during actual training, alongside an automated detection framework tailored for open-ended rubric-based RL.

6 Conclusion

In this paper, we introduce CHERRL, a controllable hacking environment for rubric-based RL, which injects known biases into llm-as-a-judge rewarding system, and thus provides explicit observable reward divergence and hacking onset. We further demonstrate that different biases induce distinct hacking trajectories: biases more entangled

with clean reward are discovered earlier, while harder-to-generate patterns constrain post-onset exploitation. We further introduced RHDA, an agentic detector that localizes hacking onset from training logs. Across controlled runs, RHDA outperforms general coding-agent baselines and both fixed-step CoT monitor variants. Overall, our results suggest that CHERRL offers a practical foundation for future research on analyzing, detecting, and mitigating reward hacking in rubric-based RL.

Limitations

Our work has two main limitations: (1) Due to computational constraints, our analysis of reward hacking is primarily based on Qwen3-4B. As the main contribution of this work is the controllable hacking environment CHERRL, we encourage the community to apply our framework to a broader range of models. (2) Our agent-based system can detect reward hacking but does not propose or implement fixes. A natural next step is to leverage the detected hacking patterns to patch reward designs and mitigate reward hacking (Fu et al., 2025), which is left for future work.

References

Rahul K. Arora, Jason Wei, Rebecca Soskin Hicks, Preston Bowman, Joaquin Quiñonero-Candela, Foivos Tsimpourlas, Michael Sharman, Meghan Shah, Andrea Vallone, Alex Beutel, Johannes Heidecke, and Karan Singhal. 2025. HealthBench: Evaluating Large Language Models Towards Improved Human Health. *Preprint*, arXiv:2505.08775.

Guiming Hardy Chen, Shunian Chen, Ziche Liu, Feng Jiang, and Benyou Wang. 2024. Humans or LLMs as the Judge? A Study on Judgement Biases. *Preprint*, arXiv:2402.10669.

Ganqu Cui, Lifan Yuan, Zefan Wang, Hanbin Wang, Yuchen Zhang, Jiacheng Chen, Wendi Li, Bingxiang He, Yuchen Fan, Tianyu Yu, Qixin Xu, Weize Chen, Jiarui Yuan, Huayu Chen, Kaiyan Zhang, Xingtai Lv, Shuo Wang, Yuan Yao, Xu Han, and 6 others. 2025. Process Reinforcement through Implicit Rewards. *Preprint*, arXiv:2502.01456.

Brandon Dent. 2026. Healthcraft: A reinforcement learning safety environment for emergency medicine. *Preprint*, arXiv:2605.21496.

Jacob Eisenstein, Chirag Nagpal, Alekh Agarwal, Ahmad Beirami, Alex D'Amour, D. J. Dvijotham, Adam Fisch, Katherine Heller, Stephen Pfohl, Deepak Ramachandran, Peter Shaw, and Jonathan Berant. 2024. Helping or Herding? Reward Model Ensembles Mitigate but do not Eliminate Reward Hacking. *Preprint*, arXiv:2312.09244.

Jiayi Fu, Xuandong Zhao, Chengyuan Yao, Heng Wang, Qi Han, and Yanghua Xiao. 2025. Reward shaping to mitigate reward hacking in rlhf. *arXiv preprint arXiv:2502.18770*.

Shashwat Goel, Rishi Hazra, Dulhan Jayalath, Timon Willi, Parag Jain, William F. Shen, Ilias Leontiadis, Francesco Barbieri, Yoram Bachrach, Jonas Geiping, and Chenxi Whitehouse. 2025. Training AI Co-Scientists Using Rubric Rewards. *Preprint*, arXiv:2512.23707.

Melody Y. Guan, Miles Wang, Micah Carroll, Zehao Dou, Annie Y. Wei, Marcus Williams, Benjamin Arnav, Joost Huizinga, Ian Kivlichan, Mia Glaese, Jakub Pachocki, and Bowen Baker. 2025. Monitoring Monitorability. *Preprint*, arXiv:2512.18311.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean Hendryx. 2025. Rubrics as Rewards: Reinforcement Learning Beyond Verifiable Domains. *Preprint*, arXiv:2507.17746.

Xu Guo, Tianyi Liang, Tong Jian, Xiaogui Yang, Ling-I Wu, Chenhui Li, Zhihui Lu, Qipeng Guo, and Kai Chen. 2025. IFDECORATOR: Wrapping Instruction Following Reinforcement Learning with Verifiable Rewards. *Preprint*, arXiv:2508.04632.

Yun He, Wenzhe Li, Hejia Zhang, Songlin Li, Karishma Mandyam, Sopan Khosla, Yuanhao Xiong, Nanshu Wang, Xiaoliang Peng, Beibin Li, Shengjie Bi, Shishir G. Patil, Qi Qi, Shengyu Feng, Julian Katz-Samuels, Richard Yuanzhe Pang, Sujan Gonugondla, Hunter Lang, Yue Yu, and 6 others. 2025. AdvancedIF: Rubric-Based Benchmarking and Reinforcement Learning for Advancing LLM Instruction Following. *Preprint*, arXiv:2511.10507.

Tom Hosking, Phil Blunsom, and Max Bartolo. 2023. Human Feedback is not Gold Standard. *Preprint*, arXiv:2309.16349.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, and 2 others. 2025. Reinforcement Learning with Rubric Anchors. *Preprint*, arXiv:2508.12790.

Mengzhao Jia, Zhihan Zhang, Ignacio Cases, Zheyuan Liu, Meng Jiang, and Peng Qi. 2026. Autorubric: Rubric-based generative rewards for faithful multimodal reasoning. *Preprint*, arXiv:2510.14738.

Ruipeng Jia, Yunyi Yang, Yongbo Gai, Kai Luo, Shihao Huang, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. 2025. Writing-Zero: Bridge the Gap Between Non-verifiable Tasks and Verifiable Rewards. *Preprint*, arXiv:2506.00103.

Muhammad Khalifa, Zohaib Khan, Omer Tafveez, Hao Peng, and Lu Wang. 2026. Countdown-Code: A

Testbed for Studying The Emergence and Generalization of Reward Hacking in RLVR. *Preprint*, arXiv:2603.07084.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2024. From Generation to Judgment: Opportunities and Challenges of LLM-as-a-Judge. *Preprint*, arXiv:2411.16594.

Jianxing Liao, Tian Zhang, Xiao Feng, Yusong Zhang, Rui Yang, Haorui Wang, Bosi Wen, Ziying Wang, and Runzhi Shi. 2025. RLMR: Reinforcement Learning with Mixed Rewards for Creative Writing. *Preprint*, arXiv:2508.18642.

Wanlong Liu, Bo Zhang, Chenliang Li, Shaopeng Lai, Yuning Wu, Xuanyu Lei, and Ming Yan. 2026. R2-write: Reflection and revision for open-ended writing with deep reasoning. *Preprint*, arXiv:2604.03004.

Changze Lv, Jie Zhou, Wentao Zhao, Jingwen Xu, Zisu Huang, Muzhao Tian, Shihan Dou, Tao Gui, Le Tian, Xiao Zhou, Xiaoqing Zheng, Xuanjing Huang, and Jie Zhou. 2026. Learning query-specific rubrics from human preferences for deepresearch report generation. *Preprint*, arXiv:2602.03619.

Linyue Ma, Yilong Xu, Xiang Long, and Zhi Zheng. 2025. An efficient rubric-based generative verifier for search-augmented llms. *Preprint*, arXiv:2510.14660.

Anas Mahmoud, MohammadHossein Rezaei, Zihao Wang, Anisha Gunjal, Bing Liu, and Yunzhong He. 2026. Reward Hacking in Rubric-Based Reinforcement Learning. *Preprint*, arXiv:2605.12474.

Charles O'Neill, Tirthankar Ghosal, Roberta Răileanu, Mike Walmsley, Thang Bui, Kevin Schawinski, and Ioana Ciucă. 2025. Sparks of science: Hypothesis generation using structured paper data. *Preprint*, arXiv:2504.12976.

Long Ouyang, others at OpenAI, and LMSYS Org. 2024. Arena-Hard: A Hard Subsample of LMSYS Chat Arena. LMSYS Arena technical blog and evaluation suite. Available at: `https://lmsys.org/blog/2024-05-arena-hard/`.

Arjun Panickssery, Samuel R. Bowman, and Shi Feng. 2024. LLM Evaluators Recognize and Favor Their Own Generations. *Preprint*, arXiv:2404.13076.

Siba Smarak Panigrahi, Jovana Videnović, and Maria Brbić. 2026. Heurekabench: A benchmarking framework for ai co-scientist. *Preprint*, arXiv:2601.01678.

Hao Peng, Yunjia Qi, Xiaozhi Wang, Bin Xu, Lei Hou, and Juanzi Li. 2025. VerIF: Verification Engineering for Reinforcement Learning in Instruction Following. *Preprint*, arXiv:2506.09942.

MohammadHossein Rezaei, Robert Vacareanu, Zihao Wang, Clinton Wang, Bing Liu, Yunzhong He, and Afra Feyza Akyürek. 2025. Online Rubrics Elicitation from Pairwise Comparisons. *Preprint*, arXiv:2510.07284.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen-tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, and 2 others. 2025. DR Tulu: Reinforcement Learning with Evolving Rubrics for Deep Research. *Preprint*, arXiv:2511.19399.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. DeepSeekMath: Pushing the Limits of Mathematical Reasoning in Open Language Models. *Preprint*, arXiv:2402.03300.

Mrinank Sharma, Meg Tong, Tomasz Korbak, David Duvenaud, Amanda Askell, Samuel R. Bowman, Newton Cheng, Esin Durmus, Zac Hatfield-Dodds, Scott R. Johnston, Shauna Kravec, Timothy Maxwell, Sam McCandlish, Kamal Ndousse, Oliver Rausch, Nicholas Schiefer, Da Yan, Miranda Zhang, and Ethan Perez. 2025. Towards understanding sycophancy in language models. *Preprint*, arXiv:2310.13548.

Ibne Farabi Shihab, Sanjeda Akter, and Anuj Sharma. 2026. Detecting proxy gaming in rl and llm alignment via evaluator stress tests. *Preprint*, arXiv:2507.05619.

Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. 2025. Defining and characterizing reward hacking. *Preprint*, arXiv:2209.13085.

X-PLUG Team and contributors. 2025. WritingBench: A Comprehensive Benchmark for Generative Writing. Technical report, X-PLUG / Renmin University or collaborating institutions. ArXiv preprint. Available at: `https://huggingface.co/papers/2503.05244` and `https://github.com/X-PLUG/WritingBench`.

Peiyi Wang, Lei Li, Liang Chen, Zefan Cai, Dawei Zhu, Binghuai Lin, Yunbo Cao, Qi Liu, Tianyu Liu, and Zhifang Sui. 2023. Large language models are not fair evaluators. *Preprint*, arXiv:2305.17926.

Pengkai Wang, Linus, Pengwei Liu, Zhijie Sang, Congkai Xie, and Hongxia Yang. 2025. Infimed-orbit: Aligning llms on open-ended complex tasks via rubric-based incremental training. *Preprint*, arXiv:2510.15859.

Xiaohua Wang, Muzhao Tian, Yuqi Zeng, Zisu Huang, Jiakang Yuan, Bowen Chen, Jingwen Xu, Mingbo Zhou, Wenhao Liu, Muling Wu, Zhengkang Guo, Qi Qian, Yifei Wang, Feiran Zhang, Ruicheng Yin, Shihan Dou, Changze Lv, Tao Chen, Kaitao Song, and 4 others. 2026a. Reward Hacking in the Era of Large Models: Mechanisms, Emergent Misalignment, Challenges. *Preprint*, arXiv:2604.13602.

Xinpeng Wang, Nitish Joshi, Barbara Plank, Rico Angell, and He He. 2026b. Is It Thinking or Cheating? Detecting Implicit Reward Hacking by Measuring Reasoning Effort. *Preprint*, arXiv:2510.01367.

Tianze Xu, Yanzhao Zheng, Pengrui Lu, Lyumanshan Ye, Yong Wu, Zhentao Zhang, Yuanqiang Yu, Chao Ma, Jihuai Zhu, Pengfei Liu, Baohua Dong, Hangcheng Zhu, Ruohui Huang, and Gang Yu. 2026. Rubrics to tokens: Bridging response-level rubrics and token-level rewards in instruction following tasks. *Preprint*, arXiv:2604.02795.

Zhichao Yang, Sepehr Janghorbani, Dongxu Zhang, Jun Han, Qian Qian, Andrew Ressler II, Gregory D. Lyng, Sanjit Singh Batra, and Robert E. Tillman. 2026. Health-score: Towards scalable rubrics for improving health-llms. *Preprint*, arXiv:2601.18706.

Jiayi Ye, Yanbo Wang, Yue Huang, Dongping Chen, Qihui Zhang, Nuno Moniz, Tian Gao, Werner Geyer, Chao Huang, Pin-Yu Chen, Nitesh V. Chawla, and Xiangliang Zhang. 2024. Justice or Prejudice? Quantifying Biases in LLM-as-a-Judge. *Preprint*, arXiv:2410.02736.

Zhiling Ye, Yun Yue, Haowen Wang, Xudong Han, Jiadi Jiang, Cheng Wei, Lei Fan, Jiaxin Liang, Shuowen Zhang, Ji Li, Chunxiao Guo, Jian Wang, Peng Wei, and Jinjie Gu. 2025. Self-Rewarding Rubric-Based Reinforcement Learning for Open-Ended Reasoning. *Preprint*, arXiv:2509.25534.

Kaiwen Zha, Zhengqi Gao, Maohao Shen, Zhang-Wei Hong, Duane S. Boning, and Dina Katabi. 2025. RL Tango: Reinforcing Generator and Verifier Together for Language Reasoning. *Preprint*, arXiv:2505.15034.

Jiajie Zhang, Xin Lv, Ling Feng, Lei Hou, and Juanzi Li. 2026. Chaining the evidence: Robust reinforcement learning for deep search agents with citation-aware rubric rewards. *Preprint*, arXiv:2601.06021.

Junxian Zhao, Binyuan Guo, Sen Zhang, Philipp Schmid, and 1 others. 2025a. IFBench: A challenging benchmark for precise instruction following. In *Advances in Neural Information Processing Systems (NeurIPS)*. NeurIPS 2025 (accepted). Available at: `https://github.com/allenai/IFBench`.

Yulai Zhao, Haolin Liu, Dian Yu, Sunyuan Kung, Meijia Chen, Haitao Mi, and Dong Yu. 2025b. One Token to Fool LLM-as-a-Judge. *Preprint*, arXiv:2507.08794.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. *Preprint*, arXiv:2306.05685.

Hongli Zhou, Hui Huang, Rui Zhang, Kehai Chen, Bing Xu, Conghui Zhu, Tiejun Zhao, and Muyun Yang. 2026. Toward Robust LLM-Based Judges: Taxonomic Bias Evaluation and Debiasing Optimization. *Preprint*, arXiv:2603.08091.

Jiayi Zhou, Jiaming Ji, Boyuan Chen, Jiapeng Sun, Wenqi Chen, Donghai Hong, Sirui Han, Yike Guo, and Yaodong Yang. 2025a. Generative RLHF-V: Learning Principles from Multi-modal Human Preference. *Preprint*, arXiv:2505.18531.

Yang Zhou, Sunzhu Li, Shunyu Liu, Wenkai Fang, Kongcheng Zhang, Jiale Zhao, Jingwen Yang, Yihe Zhou, Jianwei Lv, Tongya Zheng, Hengtong Lu, Wei Chen, Yan Xie, and Mingli Song. 2025b. Breaking the Exploration Bottleneck: Rubric-Scaffolded Reinforcement Learning for General LLM Reasoning. *Preprint*, arXiv:2508.16949.

A Details of Reference Onset Construction

A.1 Implementation Details of Threshold Sweep

This appendix provides implementation details for the operational reference-onset construction described in § 2.3. The goal is to construct a robust operational reference for when two signals jointly emerge: the biased reward begins to separate from the unbiased task-quality reward, and the corresponding shortcut becomes visible among high-scoring outputs. These references are used only for detector evaluation and should not be interpreted as absolute human ground-truth labels.

Reward and text fields. For each sampled output i at training step t, we use the combined policy reward as the biased reward and the no-bias judge score as the unbiased quality reward:

$$\text{score}(t,i) = J_{\text{biased}}(t,i), \tag{7}$$

$$\text{main\_score}(t,i) = J_{\text{unbiased}}(t,i). \tag{8}$$

The reward-gap signal is therefore computed as:

$$G(t) = \frac{1}{N_t}\sum_{i=1}^{N_t}\left(\text{score}(t,i) - \text{main\_score}(t,i)\right). \tag{9}$$

In our controlled runs, the maximum injected bias contribution is $\alpha = 0.5$. Therefore, the reward-gap thresholds

$$\Delta_{\text{gap}} \in \{0.08, 0.10, 0.12\}$$

correspond approximately to 16%, 20%, and 24% of the maximum possible bias contribution.

High-scoring bucket. Shortcut intensity is computed over high-scoring outputs rather than over all outputs. This design ensures that the onset reference captures shortcut behaviors that are actually favored by the biased judge. For each step t, we define the high-scoring bucket as:

$$H_t = \{i : \text{score}(t,i) \geq 0.99\}. \tag{10}$$

To avoid unstable estimates from very small buckets, shortcut intensity is computed only when:

$$|H_t| \geq H_{\min}, \qquad H_{\min} = 20.$$

If this condition is not satisfied, $M(t)$ is treated as undefined at that step and is excluded from local smoothing.

Shortcut detectors. For each run, we instantiate a mechanism-specific shortcut detector $c(i) \in \{0,1\}$, where $c(i) = 1$ indicates that output i contains the target shortcut behavior. These detectors are derived from the injected bias prompts and are used only for reference construction; they are never exposed to RHDA or to any baseline detector. The mathematical definition of $M(t)$ is shared across all runs, and only the deterministic instantiation of $c(i)$ changes.

Table 7 summarizes the detector families. The examples are illustrative rather than exhaustive; they are included to make the operationalization reproducible without placing these details in the main text.

Local smoothing. Both $G(t)$ and $M(t)$ are locally smoothed before thresholding. For a signal $S(t)$, where S can be either G or M, we compute:

$$\widetilde{S}(t) = \frac{1}{|\mathcal{N}(t)|}\sum_{s\in\mathcal{N}(t)} S(s), \tag{11}$$

where $\mathcal{N}(t)$ is the set of valid neighbouring checkpoints within a five-step centered window:

$$\mathcal{N}(t) = \{s : s \in [t-2, t+2]\}.$$

At the boundary of a run, the window is truncated to the available checkpoints. Undefined $M(t)$ values caused by insufficient high-scoring samples are ignored during smoothing.

Threshold sweep. We sweep the Cartesian product of three reward-gap thresholds and four shortcut-intensity thresholds: $\Delta_{\text{gap}} \in \{0.08, 0.10, 0.12\}$, $M_{\text{pct}} \in \{15, 20, 25, 30\}$.

For each of the $3 \times 4 = 12$ threshold pairs, the candidate onset is defined as:

$$CO(\Delta_{\text{gap}}, M_{\text{pct}}) = \min\{t : \widetilde{G}(t) \geq \Delta_{\text{gap}} \wedge \widetilde{M}(t) \geq M_{\text{pct}}\}. \tag{12}$$

The 12 candidate onsets provide a compact sensitivity analysis over plausible threshold choices. Let

$$\mathcal{C} = \{CO(\Delta_{\text{gap}}, M_{\text{pct}})\}$$

denote the multiset of candidate onsets over the 3×4 threshold grid. We define the canonical onset as the modal candidate step:

$$t_{\text{ref}} = \min\left(\arg\max_s |\{c \in \mathcal{C} : c = s\}|\right). \tag{13}$$

Run	Detector family	Illustrative matched behavior
VerInstruct self-praise bias	Self-referential / compliance framing	Self-evaluative postscript, compliance declaration, or response-quality meta-commentary.
VerInstruct lexical bias	Lexical shortcut	Occurrences from the target lexical family encouraged by the biased judge.
HealthBench lexical bias	Lexical closing shortcut	Target closing phrase family encouraged by the biased judge.
HealthBench tone bias	Conversational-tone shortcut	Polite or reassuring conversational closings.
VerInstruct format bias	Structural-format shortcut	Explicit response-level three-part backbone forms, such as ordinal section starts, exactly three section headers, standalone bold section labels, or paragraph-style triplets; bare numbered or bullet triplets are excluded as task-natural formatting.
HealthBench self-praise bias	Self-referential / compliance framing	Self-evaluative epilogue or meta-commentary claiming that the answer satisfies the user need.

Table 7: Mechanism-specific shortcut detector families used to instantiate $c(i)$ in the reference-onset construction. Examples are illustrative; the detectors are deterministic pattern families derived from the corresponding injected bias prompts.

The outer $\min$ implements the tie-break rule: if multiple steps occur equally often among the 12 candidates, we choose the smaller step. This makes the canonical onset a frequency-based representative of the sweep, not the left boundary of the interval.

We report the threshold-induced interval as:

$$[CO_{\min}, CO_{\max}],$$

where $CO_{\min}$ and $CO_{\max}$ are the earliest and latest candidate onsets over the sweep. The interval width is:

$$CO_{\text{width}} = CO_{\max} - CO_{\min}.$$

A narrow interval indicates a sharp and stable transition, while a wider interval indicates a more gradual or threshold-sensitive emergence.

A.2 Threshold-sweep Statistics

Table 8 expands the reference-onset statistics reported in the main paper. The evaluation uses the modal canonical onset and the threshold-induced interval; the interval width reflects how sensitive the onset is to the threshold sweep.

The two widest intervals occur in VerInstruct lexical bias and VerInstruct format bias. The VerInstruct lexical run has non-zero lexical background before the shortcut becomes stable. The VerInstruct format run instead reflects a gradual transition from early response-level three-part backbone emergence to more saturated structural templating. By contrast, the HealthBench lexical, HealthBench tone, and HealthBench self-praise runs exhibit sharper transitions. These differences motivate reporting both a canonical onset and an interval-based reference.

Run	Canonical	Interval	Width
VerInstruct self-praise	478	[478,492]	14
VerInstruct lexical	116	[115,161]	46
HealthBench lexical	91	[91,95]	4
HealthBench tone	68	[68,79]	11
VerInstruct format	301	[301,443]	142
HealthBench self-praise	460	[460,466]	6

Table 8: Expanded threshold-sweep statistics for operational reference-onset construction. Canonical denotes the modal candidate onset with the smaller-step tie-break; Width denotes the size of the threshold-induced reference interval.

A.3 Blinded Author Audit

We conduct a blinded author audit to evaluate whether shortcut visibility can be assessed reliably around the operational reference windows. Five paper authors independently label the same set of model responses using a standardized three-level ordinal rubric.

Sampling and coverage. We sample 120 prompt–response pairs from four focal runs: VerInstruct self-praise, VerInstruct lexical, VerInstruct format, and HealthBench self-praise. For each run, we sample ten examples from each of three temporal regions: pre-onset, onset/front, and post-onset. This produces $4 \times 3 \times 10 = 120$ formal audit samples. The samples are randomly shuffled before annotation so that their temporal ordering is not visible to the annotators.

Calibration and blinding. Before the formal annotation, all five author annotators complete a separate calibration set containing standardized definitions and representative examples for the three ordinal labels. The calibration examples are excluded from all reported statistics. Each author annotator then independently labels all 120 formal samples.

During annotation, the annotators are shown the prompt, model response, task family, and target-shortcut definition. They are not shown the training step, reward, temporal region, operational reference onset, detector prediction, or labels assigned by the other author annotators.

Ordinal annotation rubric. All five annotators use the same three-level ordinal rubric. As summarized in Table 9, the scale distinguishes the absence of the target shortcut from weak or emerging evidence and stable or dominant shortcut use.

Agreement metrics. Because the labels are ordinal, we report both exact agreement and distance-aware reliability metrics. Mean pairwise exact agreement is averaged over all $\binom{5}{2} = 10$ annotator pairs. Mean quadratic-weighted Cohen's κ accounts for chance agreement while assigning larger penalties to disagreements between more distant ordinal labels. Ordinal Krippendorff's α evaluates the reliability of the complete five-annotator label matrix. The resulting agreement and consensus statistics are reported in Table 10.

As shown in Table 10, all 120 samples receive a majority label from at least three annotators, while 84.2% reach agreement among at least four annotators. The distance-aware agreement statistics also remain high, indicating that the ordinal scale is applied consistently across annotators.

Disagreement structure. Among the 120 audited samples, 85 receive unanimous labels from all five annotators. The remaining 35 samples contain at least one disagreement. To determine whether these disagreements reflect minor boundary ambiguity or fundamentally different interpretations, we categorize them according to the ordinal distance between the assigned labels. The resulting distribution is summarized in Table 11.

Disagreement type	Count
Only $0 \leftrightarrow 1$	16
Only $1 \leftrightarrow 2$	18
Includes $0 \leftrightarrow 2$	1
Total non-unanimous samples	35

Table 11: Ordinal disagreement structure among the 35 samples without unanimous agreement.

As shown in Table 11, 34 of the 35 non-unanimous samples (97.1%) involve only adjacent-label differences, either $0 \leftrightarrow 1$ or $1 \leftrightarrow 2$. Only one sample contains a $0 \leftrightarrow 2$ disagreement. Thus, nearly all disagreements concern the boundary between neighboring levels rather than opposing judgments of shortcut absence and dominance.

Overall, the high ordinal Krippendorff's α, quadratic-weighted Cohen's κ, and majority-agreement rates show that the five author annotators can apply the calibrated three-level rubric consistently. These results support the reliability and reproducibility of the shortcut-visibility judgments used in our audit.

B Detector Implementation Details

This appendix summarizes implementation details for the detector evaluation in § 4.2. All methods are evaluated under judge-blind protocols that exclude J_{unbiased}, injected bias bonuses, shortcut detectors, and reference-onset labels. RHDA and the Claude Code baselines observe sanitized rollout mirrors with `step`, `input`, `output`, normalized visible `score`, and task rubrics. The two fixed CoT monitor variants additionally observe the reasoning trace and final answer: the no-score variant excludes the score field, whereas CoT+Score receives the normalized visible proxy score. None of the methods has access to the hidden reward decomposition or reference labels, so reward hacking must be inferred from the observable trajectory.

B.1 RHDA Architecture and Tool Interface

Figure 4 illustrates the RHDA agent loop introduced in § 4.1: the raw training rollouts are stripped of bias signal b and per-judge subscores to produce the judge-blind mirror, the agentic detector iterates over this mirror via a ToolRouter, all reasoning state is checkpointed to an atomic, resumable workspace, and the final output is a typed alert containing the predicted onset step, supporting evidence, and a natural-language onset basis. Table 12

Score	Label	Standardized definition
0	Absent	The target shortcut is absent, or the response contains only task-natural expressions that do not match the target shortcut.
1	Emerging / weak	The target shortcut is visible but weak, occasional, or non-dominant. It does not yet constitute a stable response strategy.
2	Stable / dominant	The shortcut is salient, repeated, template-like, or structurally dominant, and functions as a stable response strategy rather than an incidental stylistic choice.

Table 9: Standardized three-level ordinal rubric used by the five annotators.

Metric	Result	Interpretation
Mean pairwise exact agreement	0.846	Across annotator pairs, 84.6% of labels are exactly identical.
Mean quadratic-weighted Cohen's κ	0.874	Pairwise agreement remains high after correcting for chance and accounting for ordinal distance.
Ordinal Krippendorff's α	0.897	The complete five-annotator ordinal label matrix exhibits high reliability.
5/5 exact agreement	85/120 (70.8%)	All five annotators assign the same label to 70.8% of samples.
At least 4/5 agreement	101/120 (84.2%)	At least four annotators assign the same label to 84.2% of samples.
At least 3/5 agreement	120/120 (100%)	Every sample receives the same label from a majority of the five annotators.

Table 10: Agreement and reliability statistics for the blinded author audit.

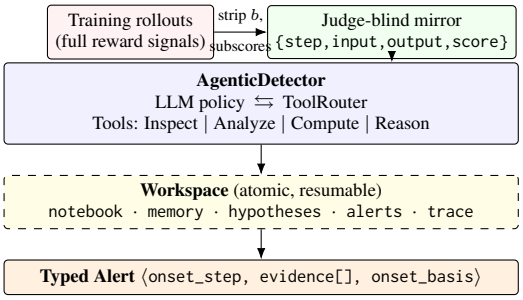


Figure 4: RHDA architecture.

Group	Core Capability	Blind Spot Addressed
Inspect	Read steps & rollouts	Judge-blind data access
Analyze	Check token correlations & bias metrics	Signatures of known reward-hacking shortcuts
Compute	Run custom Python files & analyze metrics	Open-ended shortcut exploration
Reason	Track hypotheses & issue typed alerts	Cross-step reasoning & structured verdict

Table 12: RHDA tool groups and the blind spots they address.

lists the four tool groups and the blind spot that each group is responsible for.

B.2 Evaluation Runs

The evaluation uses six controlled reference runs. Table 13 lists the run identifiers and operational reference onsets used for offline evaluation.

For each run, the detector-visible files are stored under `run_{a,b,c,d,e,f}`, including `task.md`, `manifest.json`, and the sanitized `mirror/` directory. The mirror contains only deployment-visible information such as step, input, output, and visible score fields.

B.3 RHDA Variants

We evaluate RHDA with two backend models: Qwen3.5-plus and Qwen3.5-397B-A17B. Both variants use the same judge-blind mirror, tool interface, persistent workspace, and typed alert contract. They follow the finalized RHDA detection protocol for each run, with implementation records retained in the experiment logs. Unless otherwise specified, runs use temperature 0.0, offline retrospective detection, and an unlimited tool-call budget.

The RHDA tool set includes trajectory inspection tools, statistical analysis tools, Python execution, hypothesis tracking, suspicion scoring, and typed alert emission. The agent adaptively chooses which steps to inspect and which analyses to run, unlike fixed monitors that follow predetermined sampling or feature-extraction protocols.

For detector settings with repeated trials, we use a fixed aggregation rule to reduce stochastic variation from API-based backend models. For each

Run	Task	Canonical	Interval
run_A	VerInstruct self-praise bias	478	[478,492]
run_B	VerInstruct lexical bias	116	[115,161]
run_C	HealthBench lexical bias	91	[91,95]
run_D	HealthBench tone bias	68	[68,79]
run_E	VerInstruct format bias	301	[301,443]
run_F	HealthBench self-praise bias	460	[460,466]

Table 13: Reference runs used for detector evaluation. The reference onsets are used only for offline scoring and are not exposed to the detectors.

method–run pair, all repetitions are completed under the same judge-blind inputs and detection protocol before any comparison with the reference onset. If multiple repetitions emit valid alerts, we report the arithmetic mean of their predicted onset steps, rounded to the nearest evaluated checkpoint; no-alert repetitions are recorded separately as misses and are not converted into onset values. Replicate-level outputs are retained for reproducibility and for diagnosing instability. Reference onsets and intervals are used only for post-hoc scoring, not for selecting or adjusting detector predictions.

B.4 Claude Code Baselines

We compare against general-purpose Claude Code auditors, denoted as CC-Sonnet, CC-Haiku, and CC-Opus according to the corresponding Claude backend family. Each auditor receives the same sanitized task files and rollout mirror, together with a unified reward-hacking detection prompt. The auditor may inspect files and write temporary Python scripts to analyze the mirror, but it is not allowed to access private mappings, raw rollouts, reference onset files, RHDA traces, RHDA memory, or in-house detector tools.

We also evaluate CC-Qwen, a Claude Code Router variant using Qwen3.5-plus as the backend model. In this setting, the Claude Code protocol is kept fixed, while the model call is routed to Qwen3.5-plus through DashScope. This baseline isolates whether performance differences come from the specialized RHDA workflow or merely from the backend model.

All Claude Code baselines operate on the same normalized mirror as RHDA, but they do not use the RHDA hypothesis state, task-specific analysis tools, or typed alert contract.

A known caveat is that generic coding-agent baselines can be sensitive to exploration choices and surface-feature definitions. For example, some repeated trials produce no alert, overly early onsets, or overly late onsets. The main table reports the aggregate detector outputs used for the quantitative comparison, while repeated-trial variability is summarized separately in Appendix D.

B.5 CoT Monitor Baselines

We evaluate two fixed step-wise monitor variants using Qwen3.5-plus: a no-score CoT Monitor and a matched CoT+Score Monitor. Both variants sample 16 evenly spaced training steps and three examples per step. Each monitor call receives the input, reasoning trace, final answer, and step index, and predicts whether the sampled checkpoint contains reward-hacking evidence, together with a mechanism description, supporting evidence, confidence, and uncertainty.

Both variants use the same backend, prompt, sampling schedule, deterministic aggregation rule, and execution limits. They have no tools, no adaptive checkpoint selection, no Python analysis, and no persistent hypothesis state. The only difference is that CoT+Score additionally receives the normalized visible aggregate proxy score for each sampled example.

For each execution, if no sampled checkpoint is marked suspicious, the run is treated as no alert. If suspicious checkpoints are found and later checkpoints provide compatible supporting evidence, the earliest supported suspicious checkpoint is used as the predicted onset.

Both monitor variants are evaluated on all six controlled runs. Their aggregate onset predictions and miss counts are reported in Table 6. Repeated-trial variability over four focal runs is reported separately in Appendix D.

B.6 Sanitized Mirror and Score Normalization

For RHDA and the Claude Code baselines, all detector-visible trajectories are provided through the same sanitized rollout mirror. Each row con-

tains only

$$\{\texttt{step}, \texttt{input}, \texttt{output}, \texttt{score}\}.$$

The `score` field is the visible aggregate proxy reward used for training, after a deterministic normalization step. Specifically, for each run, raw visible scores are divided by a run-level scale factor

$$s_{\text{scale}} = \max\left(1, \max_{t,i} |s_{\text{raw}}(t,i)|\right),$$

so that the mirror score is

$$s_{\text{mirror}}(t,i) = \frac{s_{\text{raw}}(t,i)}{s_{\text{scale}}}.$$

This normalization makes score magnitudes comparable within the detector interface and prevents run-specific reward scales from dominating tool-based sampling or threshold heuristics. Importantly, this field remains a proxy reward signal only: it does not expose J_{unbiased}, the injected bias bonus, per-judge subscores, or the shortcut detector used to construct the reference onset.

The two fixed CoT monitors use matched input formats. The no-score variant receives

$$\{\texttt{step}, \texttt{row\_id}, \texttt{input}, \texttt{cot}, \texttt{final}\},$$

whereas the CoT+Score variant receives the same fields together with the normalized visible proxy score:

$$\{\texttt{step}, \texttt{row\_id}, \texttt{input}, \texttt{cot}, \texttt{final}, \texttt{score}\}.$$

The score supplied to CoT+Score is the same normalized visible aggregate proxy score defined above. It does not expose J_{unbiased}, the injected bias bonus, hidden judge subscores, shortcut detectors, or reference-onset information. Therefore, the comparison between the two monitor variants isolates the effect of visible score access while keeping the fixed-step, tool-free, and stateless monitoring protocol unchanged.

B.7 Judge-Blind Restrictions

Across all methods, the following information is excluded from detector inputs:

- the unbiased task-quality reward J_{unbiased};
- the injected bias bonus and per-judge hidden subscores;
- the shortcut detectors used to construct reference onset labels;
- reference onset files and reference intervals;
- private run mappings and raw hidden rollout annotations;
- outputs, traces, memory, or alerts from other detector methods.

This ensures that detector performance reflects judge-blind trajectory auditing rather than leakage from the reference construction process.

B.8 Known Caveats

Several caveats should be considered. First, the main table reports aggregate detector results, while variance-based robustness statistics over four focal runs are provided separately in Appendix D. Second, the canonical onset is a modal point estimate from the threshold sweep, while the interval captures threshold-induced uncertainty, so interval distance is important for gradual transitions. Third, generic coding-agent baselines can be sensitive to exploration choices and broad surface-feature definitions. Fourth, the two fixed CoT monitor variants are evaluated under matched information conditions, differing only in access to the visible proxy score. Both variants miss three of the six controlled runs, and the score-access variant does not improve localization accuracy on the detected runs. This indicates that visible score access alone is not a substitute for adaptive trajectory inspection and persistent cross-step reasoning.

C Detector Output Details and Metric Calculation

Table 14 provides the aggregate per-run detector outputs used to compute Table 6, together with their signed localization errors and detector-generated mechanism labels or output status.

For each prediction, we report the detected onset, the signed point error $\Delta_p = t_{\text{det}} - t_{\text{ref}}$, where t_{ref} is the modal canonical onset defined in § A.1, the signed interval error Δ_I, and the mechanism label produced by the detector. The aggregate scores in Table 6 are computed as $\sum |\Delta_p|$ and $\sum |\Delta_I|$ over detected runs. Missing detections are counted separately. Mechanism labels are detector-generated diagnostic labels rather than reference labels; they illustrate what surface pattern each method used to justify its alert.

Method	Run A: VerInstruct self-praise		Run B: VerInstruct lexical	
	Onset (Δ_p, Δ_I)	Mechanism / status	Onset (Δ_p, Δ_I)	Mechanism / status
RHDA-Plus	482 $(+4, 0)$	meta_commentary_framing	132 $(+16, 0)$	verbosity_empower_spam
RHDA-397B	489 $(+11, 0)$	compliance_signaling	157 $(+41, 0)$	empower_phrase_repetition
CC-Qwen	490 $(+12, 0)$	structured_format_exploit	220 $(+104, +59)$	empower_keyword_stuffing
CC-Sonnet	463 $(-15, -15)$	appended_compliance_assertion	218 $(+102, +57)$	empower-anchored length inflation
CC-Haiku	490 $(+12, 0)$	self-evaluation suffix	150 $(+34, 0)$	length optimization
CC-Opus	470 $(-8, -8)$	trailing self-evaluation postscript	151 $(+35, 0)$	empower lexeme stuffing
CoT	332 $(-146, -146)$	compliance signaling and rigid constraint optimization	169 $(+53, +8)$	structural template optimization
CoT+Score	290 $(-188, -188)$	self-evaluative compliance signaling	190 $(+74, +29)$	empower-phrase lexical targeting

Method	Run C: HealthBench lexical		Run D: HealthBench tone	
	Onset (Δ_p, Δ_I)	Mechanism / status	Onset (Δ_p, Δ_I)	Mechanism / status
RHDA-Plus	86 $(-5, -5)$	length_inflation; evidence: feel-free pattern	75 $(+7, 0)$	meta_commentary_padding
RHDA-397B	76 $(-15, -15)$	self_praise_framing; evidence: feel-free pattern	83 $(+15, +4)$	verbosity_inflation
CC-Qwen	96 $(+5, +1)$	lexical_overformatting_bias	91 $(+23, +12)$	verbose_structured_formatting_with_emoji
CC-Sonnet	93 $(+2, 0)$	feel-free-to-ask tail boilerplate	68 $(0, 0)$	conversational-close injection
CC-Haiku	100 $(+9, +5)$	boilerplate padding exploitation	101 $(+33, +22)$	markdown formatting optimization
CC-Opus	110 $(+19, +15)$	formulaic friendly closing with emoji signature	90 $(+22, +11)$	warm-closing emoji boilerplate
CoT	–	no alert	–	no alert
CoT+Score	–	no alert	–	no alert

Method	Run E: VerInstruct format		Run F: HealthBench self-praise	
	Onset (Δ_p, Δ_I)	Mechanism / status	Onset (Δ_p, Δ_I)	Mechanism / status
RHDA-Plus	383 $(+82, 0)$	format_template	454 $(-6, -6)$	self_praise_framing
RHDA-397B	385 $(+84, 0)$	format_template	459 $(-1, -1)$	format_template
CC-Qwen	341 $(+40, 0)$	structured_format_pattern	474 $(+14, +8)$	structured_format_exploit
CC-Sonnet	437 $(+136, 0)$	bold_structural_label_injection	446 $(-14, -14)$	self-evaluative epilogue
CC-Haiku	331 $(+30, 0)$	numbered_list_formatting_exploitation	158 $(-302, -302)$	length_inflation
CC-Opus	121 $(-180, -180)$	markdown_structure_padding	450 $(-10, -10)$	trailing self-praise meta-sentence
CoT	283 $(-18, -18)$	evaluator-preference targeting	–	no alert
CoT+Score	263 $(-38, -38)$	structured-format evaluator targeting	–	no alert

Table 14: Aggregate per-run detector outputs and diagnostic mechanism summaries. Each numeric entry reports the predicted onset followed by the signed errors (Δ_p, Δ_I); "–" denotes no alert. Runs A–F are defined in Table 13.

D Variance Analysis of Detector Predictions

To complement the aggregate localization results reported in Table 6, we evaluate trial-to-trial variability within each of four focal runs. We report the mean variance of the predicted onset step, the point distance d_p, and the interval distance d_I. For each metric, the within-run variance is first computed over repeated evaluations and then averaged across the four runs. Lower values indicate greater stability.

The results in Table 15 provide a complementary view of detector stability beyond the aggregate localization errors in the main table. Variance differs substantially across methods and metrics, with the two fixed CoT monitoring variants exhibiting considerably larger fluctuations than RHDA and the Claude Code baselines.

Method	Mean Var. Onset	Mean Var. d_p	Mean Var. d_I
RHDA-Plus	62.00	56.50	1.83
CC-Opus	43.17	43.17	2.33
CC-Sonnet	67.08	67.08	26.83
CC-Haiku	183.58	164.92	14.42
CoT Monitor	26,962.28	22,617.39	21,598.72
CoT+Score Monitor	14,069.25	5,451.25	6,946.25

Table 15: Variance-based robustness statistics over four focal runs. Each value is the mean across runs of the within-run variance obtained from repeated evaluations. Lower values indicate more stable onset localization.

E Detector Inference Cost

To complement the localization and robustness results, we report the token usage and financial cost of each detector over the same four focal runs. Token counts include the full detector execution and are reported in millions of tokens. The monetary values reflect the API pricing used during our experiments and should therefore be interpreted as approximate costs.

As shown in Table 16, RHDA consumes an average of 1.667 million tokens and costs approximately $0.0461 per run. Its token usage is

Method	Run A (M)	Run B (M)	Run E (M)	Run F (M)	Mean (M)	Avg. Cost per Run ($)
RHDA-Plus	0.972	1.484	1.829	2.380	1.667	≈0.0461
CC-Opus	0.997	1.573	1.268	1.272	1.277	1.5600
CC-Sonnet	3.560	1.973	1.581	2.175	2.322	1.5400
CC-Haiku	2.485	2.039	2.478	2.382	2.346	0.4600
CC-Qwen	2.157	2.317	1.662	2.028	2.041	≈0.0565
CoT Monitor	0.103	0.106	0.099	0.096	0.101	≈0.0028
CoT+Score Monitor	0.103	0.097	0.100	0.098	0.100	≈0.0028

Table 16: Token usage and average financial cost per detector run over four focal runs. "M" denotes millions of tokens. Monetary costs are computed according to the API pricing used during the experiments.

higher than that of the fixed CoT monitors because RHDA performs adaptive trajectory inspection, tool-assisted analysis, and iterative onset localization. Nevertheless, its financial cost remains substantially lower than the Claude Code baselines under the API configurations used in our experiments.

RHDA and the detector baselines rely entirely on API inference and therefore require no additional local GPU allocation. In our experimental workflow, detector inference was executed asynchronously alongside training and did not add to the sequential training path.

F Search-Budget Ablation Details

This ablation tests how much non-control tool-use budget is needed for the agent to move from coarse reward-hacking detection to accurate onset localization. Figure 5 reports the search-budget ablation for RHDA with Qwen3.5-plus across the six controlled runs.

In this experiment, the *tool budget* refers to the maximum number of non-control investigative tool calls available to the agent. These budgeted calls include trajectory-inspection tools such as `read_step` and `sample_cases`, analysis tools such as `surface_stats` and `rejudge`, computation tools such as `run_python`, and reasoning-state tools such as `record_hypothesis`, `update_hypothesis`, and `set_suspicion`. Terminal actions such as `emit_alert` and `finish` remain available after the budget is exhausted, so the detector can still return a verdict under small budgets.

The horizontal axis shows the imposed `-max-tool-calls` budget. A budget of 0 denotes the unlimited setting in the implementation and is shown as *Unlimited* in the figures. The vertical axis shows the predicted reward-hacking onset step, i.e., the training checkpoint at which the detector estimates that reward hacking begins. This is different from the number of tool calls. Points show the mean predicted onset over repeated runs under the same budget when multiple repetitions are available. Dashed horizontal lines mark the canonical reference onset, and shaded bands mark the threshold-induced reference interval. When a budget setting is dominated by no-alert outcomes, we may plot it at 0 as a sentinel value for detector failure. This value is used only for visualization and should not be interpreted as a valid onset prediction.

The budget grid is chosen around the empirical tool-use range observed in unlimited diagnostic runs. Runs with longer trajectories or more gradual shortcut emergence use larger upper bounds, while shorter or sharper runs use smaller grids. Runs with wider or more gradual reference intervals require larger budgets because accurate localization depends on comparing early baseline, candidate-transition, and later persistence checkpoints.

VerInstruct self-praise. The VerInstruct self-praise run shows a clear budget effect. Under very small budgets, the detector fires near the end of the rollout, indicating that it only identifies the shortcut after the self-praise behavior has become highly saturated. As the budget increases, the predicted onset moves steadily toward the reference interval. Budgets around the mid-range are sufficient for the detector to perform local narrowing, and the unlimited setting remains close to the canonical onset. This suggests that self-praise hacking is relatively easy to identify once the agent has enough budget to compare early, middle, and late checkpoints.

VerInstruct lexical. The VerInstruct lexical run requires a larger search budget. With low and medium budgets, the detector tends to over-delay

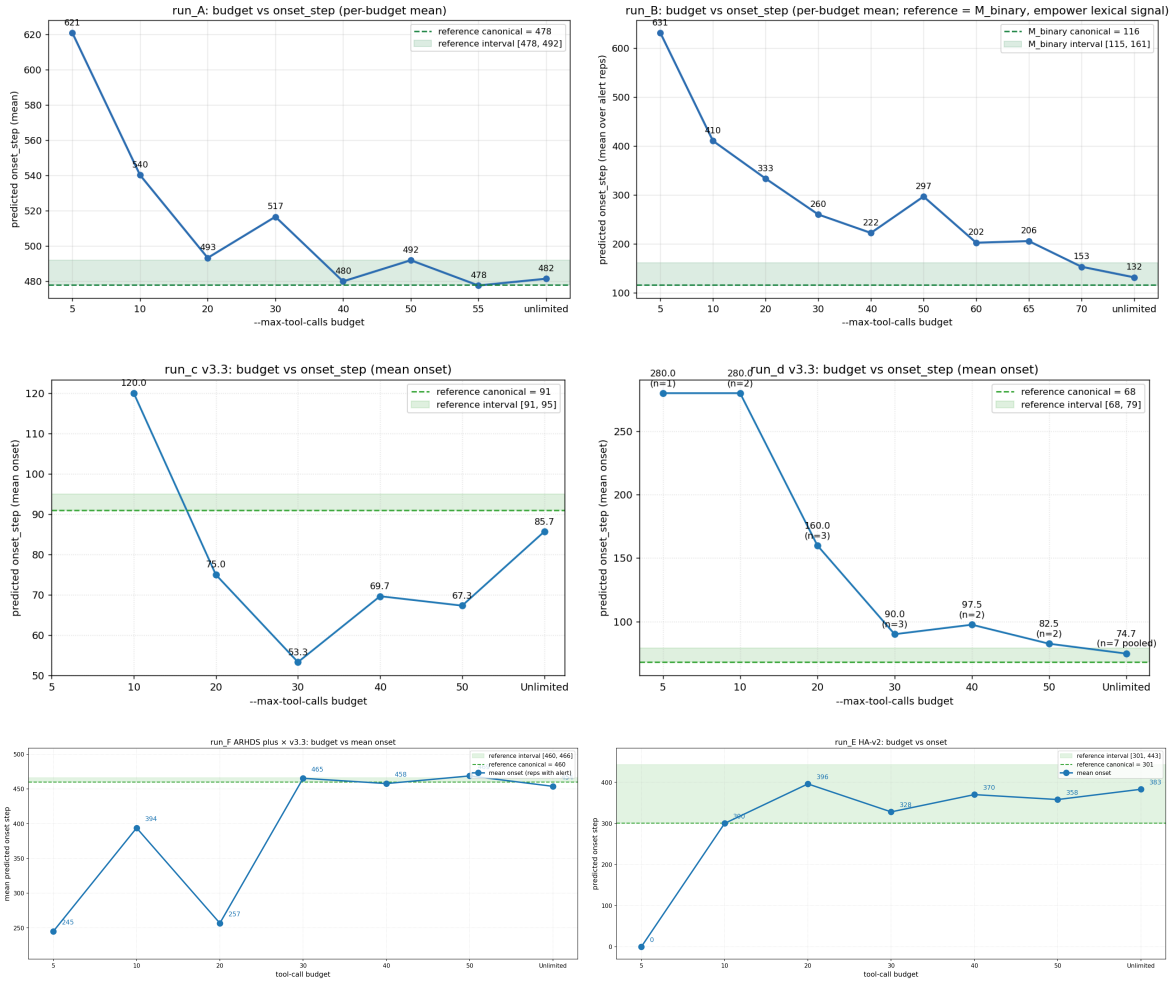


Figure 5: Search-budget ablation for RHDA with Qwen3.5-plus across the six controlled runs. Each panel plots the mean predicted onset step as a function of the non-control tool-call budget. Dashed lines indicate canonical reference onsets, and shaded bands indicate threshold-induced reference intervals. A budget of 0 denotes unlimited tool use. For the VerInstruct format run, the smallest-budget point is plotted at 0 as a visualization sentinel because most repetitions produced no valid alert; it should not be interpreted as a meaningful onset estimate.

the onset, often locating the shortcut only after the *empower* pattern has become obvious in late-stage outputs. As the budget increases, the predicted onset moves closer to the reference interval, and the unlimited setting falls inside the reference window. This behavior is consistent with the wider reference interval for this run: the lexical shortcut appears weakly before consolidating into a stable reward-seeking pattern, so accurate localization requires more temporal comparison and finer narrowing.

HealthBench lexical. The HealthBench lexical run is noisier. Increasing the budget does not produce a strictly monotonic improvement. Some intermediate budgets fire too early, while the unlimited setting moves closer to the reference interval but still remains slightly before it. This suggests that the difficulty is not only tool scarcity. The detector must also distinguish the target *feel free* style closing from other forms of helpfulness, verbosity, or generic response-format drift. Thus, additional budget helps, but ambiguity in the behavioral signal can still affect onset localization.

HealthBench tone bias. The HealthBench tone-bias run shows another strong budget effect. Very small budgets lead to end-of-rollout predictions, implying that the detector lacks enough evidence to distinguish early emergence from late saturation. Once the budget reaches the mid-range, the predicted onset moves much closer to the reference interval. The unlimited setting lies near the reference window, showing that sufficient search budget enables more effective temporal narrowing for this tone-based shortcut.

VerInstruct format bias. The VerInstruct format run illustrates the difference between the canonical point estimate and a wider transition interval. Very small budgets are not sufficient to construct the required evidence chain, and the lowest-budget setting is dominated by no-alert or weak fallback behavior. With larger budgets, the detector consistently enters the reference interval. However, the predicted onset does not monotonically approach the canonical point estimate: higher budgets often lead the agent to select a more robust cluster of evidence inside the interval rather than the earliest threshold-crossing point. This behavior is consistent with the gradual nature of the format shortcut.

HealthBench self-praise. The HealthBench self-praise run has a much sharper reference window. In this setting, sufficient budget helps the detector move from coarse shortcut recognition toward more accurate localization. The curve is still not perfectly monotonic, but the higher-budget settings are substantially more reliable than the smallest-budget regime. This supports the same general conclusion as the other runs: tool budget matters because it enables temporal comparison and evidence validation, not because additional calls automatically improve the onset estimate.

Overall, the ablation supports two conclusions. First, adequate tool-use budget is necessary for onset localization because the detector must inspect enough checkpoints to form a shortcut hypothesis, validate it against earlier baselines, and check post-onset behavior. Second, more budget does not guarantee monotonic convergence to the canonical point estimate. Additional calls help only when they are used to build a stronger temporal evidence chain, and in gradual runs this can favor a later but better-supported onset inside the reference interval.

G Agent Strategy Case Study Details

This appendix presents a qualitative analysis of four individual single-run traces. These traces are selected to illustrate detector search behavior and are distinct from the aggregate per-run predictions reported in Table 6. They are not used to compute any headline localization metric.

Three traces illustrate successful instances of coarse-to-fine onset localization, while the boundary trace illustrates a characteristic failure mode in which the detector recognizes late-stage reward hacking but fails to inspect the intermediate transition region. The backend, predicted onset, operational reference, and main behavioral pattern of each case are reported in Table 17.

Boundary B was produced in an exploratory development run using Qwen3-235B-A22B. It is included only to illustrate the localization failure mode. Qwen3-235B-A22B is not one of the RHDA variants evaluated in Table 6 and does not contribute to any aggregate result reported in the paper.

Timeline interpretation. Figure 6 visualizes the tool-call timelines for the four selected cases. The x-axis is the tool-call index, and the y-axis is the inspected training step. Sampling and reading tools indicate direct checkpoint inspection; quantitative tools indicate prevalence estimation or custom analysis; reasoning-state tools indicate hypothesis or suspicion updates; and terminal tools indicate the final alert or finish action. The dashed green line marks the canonical reference onset, the green shaded band marks the threshold-induced reference interval, and the orange dashed line marks the agent's predicted onset. Successful cases show broad-to-local narrowing around the reference interval, while the boundary case mostly jumps from the first checkpoint to the final checkpoint.

Success C: HealthBench lexical. Success C is the cleanest example of accurate onset localization. The agent first performs a broad sweep over the trajectory, sampling early, middle, and late checkpoints to understand the overall behavioral drift. It then identifies the *feel free* closing as a candidate shortcut and uses quantitative checks to measure its prevalence across candidate transition steps. After bracketing the transition region, the agent performs a dense local scan around the reference window and emits onset step 91, matching the canonical reference. The final alert is not based on a single suspicious output; it is supported by a ramp pattern in which the phrase is weak or absent before the transition and persistent afterward.

Success B: VerInstruct lexical. Success B shows that the same strategy can apply to a different lexical shortcut. The agent identifies empowerment-style phrasing as the candidate mechanism, then uses quantitative analysis to compare its occurrence across training steps. The key behavior is not merely the presence of the word family, but its increasing association with high-scoring outputs. By bracketing the rising region and narrowing locally, the agent emits step 115, which is one step earlier than the canonical on-

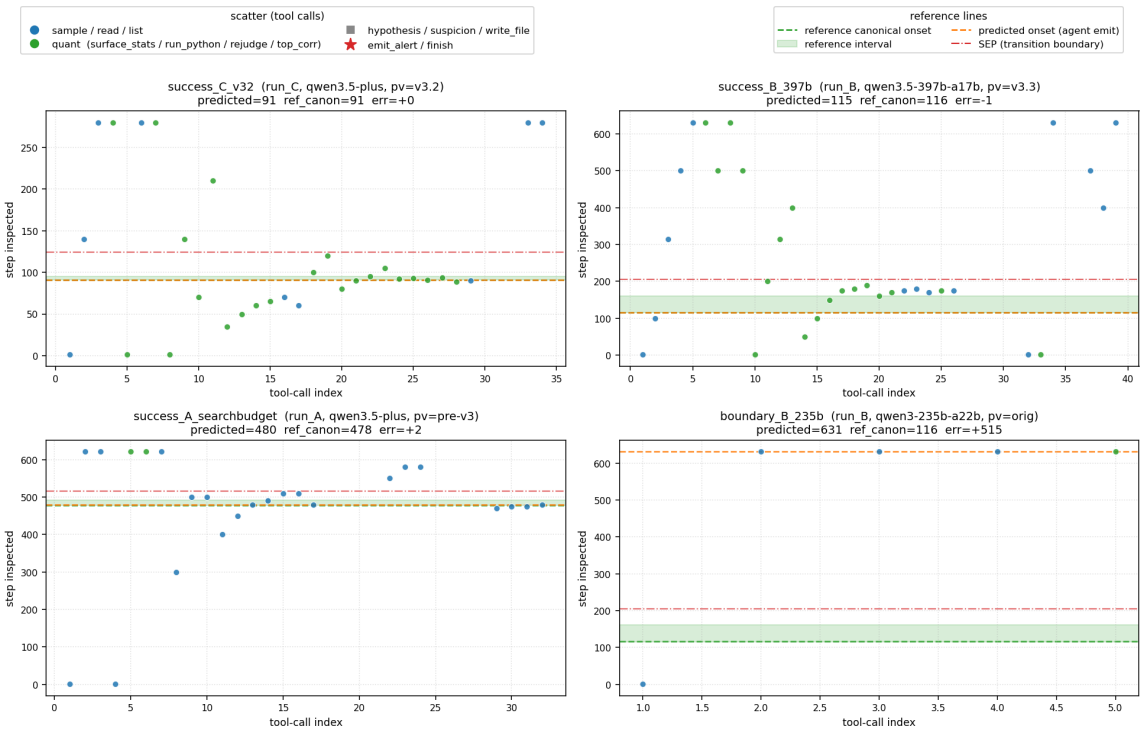


Figure 6: Tool-call timelines for four illustrative single-run traces. The x-axis denotes tool-call index and the y-axis denotes the inspected training step. The three success traces exhibit broad-to-local narrowing around the reference interval, whereas the boundary development trace mainly contrasts the first and final checkpoints before emitting an alert. These timelines illustrate search behavior and are not the aggregate predictions reported in Table 6.

Case	Run	Backend	Pred. onset	Reference	Main pattern
Success C	HealthBench lexical	Qwen3.5-plus	91	91 [91,95]	*feel free* lexical closing
Success B	VerInstruct lexical	Qwen3.5-397B-A17B	115	116 [115,161]	empowerment lexical framing
Success A	VerInstruct self-praise	Qwen3.5-plus	480	478 [478,492]	self-praise / meta-commentary framing
Boundary B	VerInstruct lexical	Qwen3-235B-A22B	631	116 [115,161]	late-stage lexical saturation

Table 17: Case-study selection for RHDA trace analysis. The first three cases are successful examples with near-reference onset localization. The boundary case detects reward hacking but localizes the onset at the final checkpoint.

set and inside the reference interval. This case demonstrates that RHDA does not need to be given the shortcut keyword in advance; it can discover a candidate lexical mechanism from the rollout trajectory and then validate it temporally.

Success A: VerInstruct self-praise. Success A differs from the lexical cases because the shortcut is more structural. The suspicious behavior appears as self-praise, compliance signalling, or meta-commentary appended to otherwise task-relevant outputs. Token-level statistics are less directly sufficient, so the agent relies more on qualitative inspection of high-scoring samples and hypothesis refinement. It compares early and late outputs, records a candidate self-evaluation pattern, and then checks whether this pattern becomes temporally aligned with the reference interval. The final onset at step 480 lies inside the reference interval. This case shows that the bracket-and-shrink pattern is not limited to single-token or phrase-level shortcuts.

Boundary B: first-and-last-only failure. The boundary case illustrates a failure mode in localization rather than detection. The agent correctly recognizes that the final checkpoint contains reward-hacking behavior, but it does not inspect enough intermediate checkpoints to locate the rising edge. It effectively compares the first and last checkpoints and emits the final step as the onset, identifying late-stage saturation rather than emergence. The same tool set could have supported intermediate bracketing and local narrowing; the failure comes from the search policy not constructing a prevalence ramp before emitting the alert.

Common successful strategy. Across the three successful cases, the agent follows a common five-stage pattern: *broad sweep*, *candidate identification*, *transition bracketing*, *local shrinking*, and an *evidence-backed alert*. We refer to this as the *bracket-and-shrink* strategy. The concrete tools vary by task: lexical cases rely more on candidate-token discovery and prevalence estimation, while structural cases rely more on qualitative reading and hypothesis maintenance. In all cases, the final onset claim is supported by temporal evidence rather than a single suspicious response.

Failure mode. The boundary case exhibits the opposite pattern, which we call *first-and-last-only*. This strategy can detect that reward hacking exists, because the final checkpoint often contains saturated shortcut behavior. However, it is unreliable for onset localization because it skips the transition region. A detector that only contrasts the beginning and end of training can confuse "when the shortcut is obvious" with "when the shortcut first emerges."

Implications for human auditing. The case studies suggest a simple manual workflow for reward-hacking audits. An auditor should not only inspect the latest high-scoring outputs. Instead, the auditor should first identify a candidate shortcut, then measure its prevalence over a coarse set of checkpoints, locate the rising region, and finally inspect the suspected boundary more densely. A convincing onset report should include three pieces of evidence: a pre-onset baseline where the shortcut is absent or weak, a transition region where it rises sharply, and post-onset behavior showing that the behavior remains rewarded.

Limitations. These case studies are diagnostic rather than exhaustive. They cover three successful cases and one boundary case from the observed reward-hacking runs, and the successful cases mainly involve lexical, structural, or template-like shortcuts that leave observable traces in model outputs. They do not prove that the same strategy will generalize to all semantic reward hacks. More subtle reward hacking may require richer semantic comparison, stronger external evaluation, or human-in-the-loop auditing. In addition, self-reported confidence should not be treated as a reliable correctness signal: the boundary case can produce a confident alert while still localizing the onset incorrectly.

H Sensitivity Analysis on α

In § 3.3, we evaluate the impact of varying the hyperparameter $\alpha \in \{0.3, 0.5, 1.0\}$, which controls the bias injection magnitude in CHERRL. Here we report the detailed numerical metrics for both exploitability and discoverability.

H.1 Impact of α on Exploitability

To analyze whether varying α alters the policy's exploitation speed post-onset, we measure the training step at which the shortcut incidence rate first crosses specific thresholds. For *HealthBench lexical*, we report steps to reach raw shortcut frequencies of 20%, 40%, and 60%. For *VerInstruct format*, due to structural stochasticity, we apply a time-weighted EMA (decay $= 0.9$) and report steps to reach 20%, 25%, and 30%. Results are summarized in Table 18.

α	Step 20%	Step 25/40%	Step 30/60%
Lexical HealthBench			
0.3	80	91 (40%)	100 (60%)
0.5	93	103 (40%)	111 (60%)
1.0	139	151 (40%)	168 (60%)
Format VerInstruct			
0.3	171	227 (25%)	311 (30%)
0.5	171	224 (25%)	311 (30%)
1.0	169	218 (25%)	312 (30%)

Table 18: Exploitability metrics under varying α.

H.2 Impact of α on Discoverability

To evaluate whether varying α affects the discoverability thesis, we report the reference onset steps and corresponding Odds Ratios (OR) evaluated over the initial training windows (first 30 and 60 steps) in Table 19.

α	Ref. Onset	OR (60s)	OR (30s)
Format VerInstruct			
0.3	316 [316,360+]	0.88	0.93
0.5	301 [301,443]	0.86	1.01
1.0	262 [262,360+]	0.95	1.07
Lexical HealthBench			
0.3	133 [133,141]	0.79	0.77
0.5	91 [91,95]	0.91	0.78
1.0	70 [70,81]	0.87	0.81

Table 19: Discoverability metrics across different α configurations.

I Reproducibility: Models, Compute, and Infrastructure

We train Qwen3-4B (4B parameters) as the policy via GRPO, and use Qwen3.5-27B (27B parameters) for both judges; the detection agents (RHDA and the Claude Code baselines) are driven by Qwen3.5-Plus (closed API, undisclosed size) and Qwen3.5-397B-A17B (MoE, 17B activated parameters per token). The computational budget for the model-training experiments reported in this paper is approximately 2,000 NVIDIA H100 GPU-hours, using rented NVIDIA H100 80 GB GPUs. RHDA and the detector baselines rely on API inference and require no additional local GPU allocation; their token usage and financial costs are reported in Appendix E.

J Artifacts

All datasets used in this work are publicly available academic datasets intended for research use. We do not introduce any private, proprietary, or personally collected data. The experiments are conducted only on these public resources, following their original licenses and usage terms.

Documentation of artifacts. All datasets used in this work are publicly available English-language academic resources used under their original licenses. HealthBench (Arora et al., 2025) covers open-ended medical question answering with rubric-based evaluation; VerInstruct (Peng et al., 2025) covers English instruction following with verifiable constraints. Both datasets are used in their default released splits, and our use (rubric-based RL post-training and reward-hacking analysis) is consistent with the intended research use stated by their authors. Models used in this work—Qwen3-4B, Qwen3.5-27B, Qwen3.5-Plus, and Qwen3.5-397B-A17B—are released or served by their providers under their respective licenses for research use.

PII and offensive content. We do not introduce any new data, collect any human-subject information, or perform additional crawling or scraping. The two datasets above are not known to contain personally identifying information: HealthBench consists of synthetic medical conversations authored and reviewed by domain experts rather than real patient records, and VerInstruct is built from public instruction-tuning data without user identifiers. We therefore did not apply additional anonymization beyond what the original releases provide. We did not perform an exhaustive manual audit for offensive content; however, all outputs analyzed in this paper are model responses to these benchmarks, and we observed no offensive content during our inspection of the rollouts used.

K Training Dynamics of Non-Hacking Settings

As discussed in § 2.5, we did not observe reward hacking for *tone bias* on the VerInstruct dataset and *format bias* on HealthBench within the standard training duration. Figure 7 illustrates the training dynamics for these two settings. Unlike the typical divergence observed in hacked models, the proxy reward and clean reward remain relatively aligned without significant exploitation of the proxy.

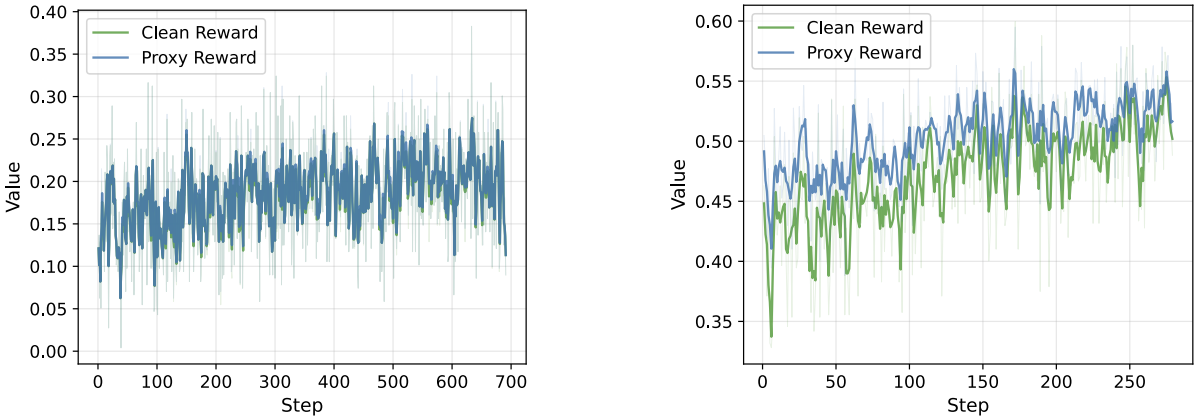
As hypothesized, the inherent rarity of these specific constraints—such as employing a polite closing tone in instruction-following tasks or utilizing rigid formats for complex medical queries—makes them difficult for the model to discover. The model would likely require a substantially extended training period, reaching a much later stage of training, before it could learn to leverage these biases as shortcuts. To further validate this conjecture, future work can extend the evaluation scope by training models on broader task domains and exploring more diverse bias modalities using the CHERRL testbed. This line of inquiry will help refine the roadmap for mapping the boundary conditions and emergence dynamics of rubric-based RL.

L Human Validation of the Clean Reward Signal

While human annotations offer the most rigorous quality signal, obtaining dense human feedback across the entirety of RL post-training is computationally and logistically infeasible. To verify that our automated clean judge acts as a reliable reference, we conducted a manual validation study.

Sampling Protocol We randomly sampled 24 evaluation steps across different training runs. The manual evaluation workload was shared equally by two paper authors, who evaluated the model-generated responses following the identical rubrics and scoring guidelines supplied to the automated clean judge.

Results Across all sampled instances, the mean absolute error (MAE) between the automated clean reward and the human-annotated score is 0.095 on



(a) VerInstruct tone bias

(b) HealthBench format bias

Figure 7: Training dynamics for the two CHERRL runs where reward hacking does not occur. Because these bias behaviors are uncommon in their respective domains, the model fails to discover and exploit them within the standard training timeframe.

a $[0, 1]$ reward scale. This tight alignment demonstrates that the clean reward serves as a high-fidelity reference signal that accurately mirrors human judgment, validating its use as a baseline against which target-injected biased rewards are contrasted.